Toward Specification and Composition of BoxScript Components

H. Conrad Cunningham, Yi Liu, and Pallavi Tadepalli

University of Mississippi, University, MS 38677

Abstract

BoxScript is a Java-based, component-oriented programming language whose design seeks to address the needs of teachers and students for a clean, simple language. This paper briefly describes BoxScript and presents the authors' preliminary ideas on specification of components and their compositions.

BoxScript

Goal: To develop a simple, Java-based, component-oriented language that enables students to "think in components" and build simple systems.

A *box* is a strongly encapsulated component with:

. Interfaces represented by *interface handles* and Java *interface types*
  - *provided interfaces* give operations available to clients
  - *required interfaces* give operations used on other boxes

. Three types
  - *abstract box* declares interfaces to be implemented by child boxes
  - *atomic box* implements provided interfaces as Java classes
  - *compound box* composes other boxes to form composite box and uses their interface implementations

```
abstract box DateAbs
{  provided interface DayCal Dc;   //Dc is handle of interface DayCal  }

abstract box CalendarAbs
{  provided interface Display Dis;
   required interface DayCal DayC;
}
```

Figure 1. Abstract boxes DateAbs **and** CalendarAbs

```
box Date implements DateAbs
{  provided interface DayCal Dc; }

box Calendar implements CalendarAbs
{  provided interface Display Dis;
   required interface DayCal DayC;
}
```

Figure 2. Atomic boxes Date **and** Calendar

Composition of boxes into a compound box

. hides all provided interfaces that are not explicitly exposed

. must expose every required interface that is not wired to a provided interface of a box

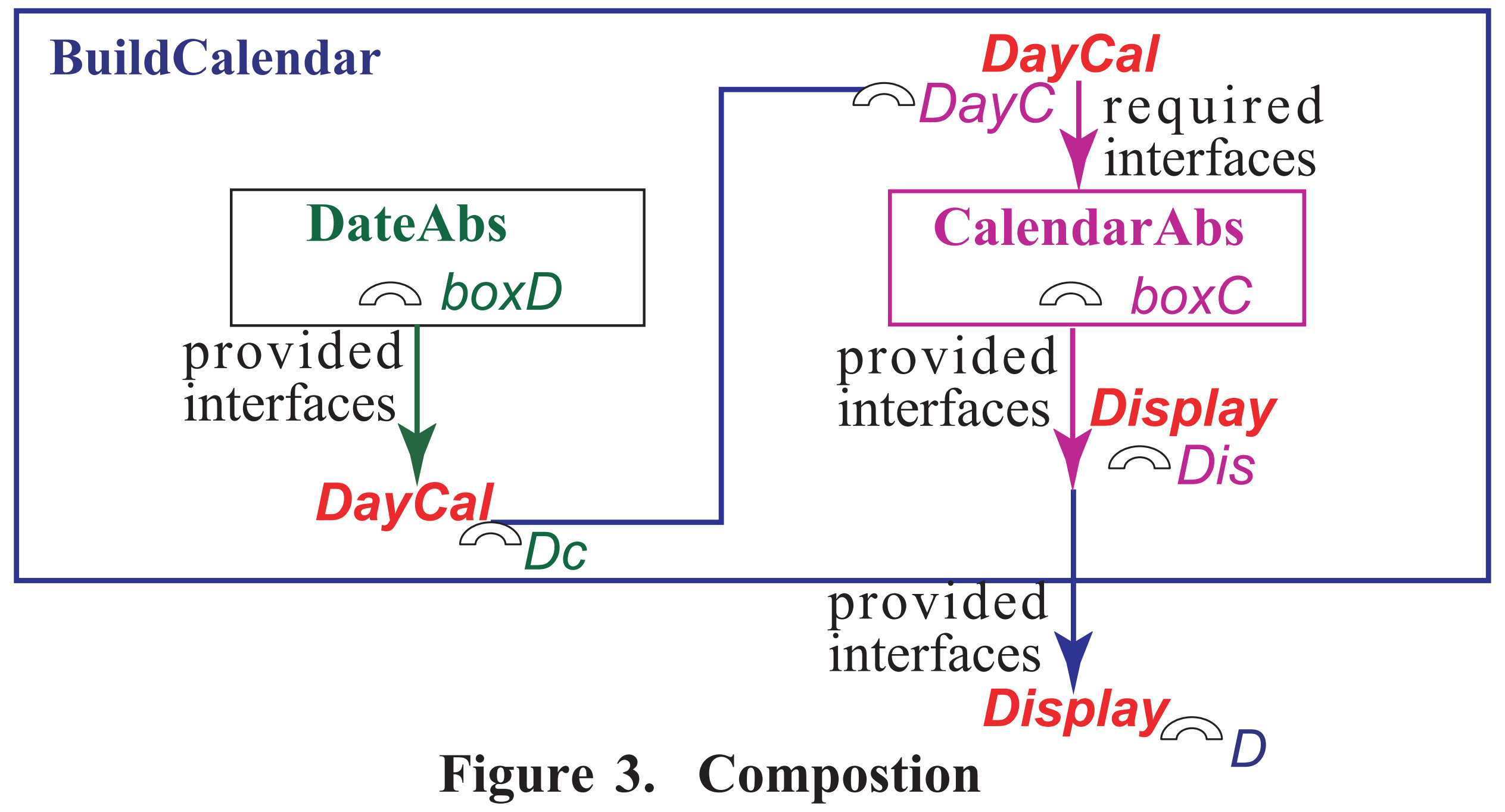

Figure 3. Compostion

```
abstract box BuildCalendarAbs
{  provided interface Display D;    }
```

Figure 4a. Abstract box BuildCalendarAbs

```
box BuildCalendar implements BuildCalendarAbs
{  composed from DateAbs boxD, CalendarAbs boxC;
   //boxD is box handle for DateAbs and boxC is box handle for CalendarAbs
   provided interface Display D from boxC.Dis;
   connect boxC.DayC to boxD.Dc;
}
```

Figure 4b. Compound box CalendarAbs

Specification

An interface *information model* consists of pair (V,I):

. V is a set of abstract variables representing state of component instance

. I is an invariant that must hold in all client-visible states

Preconditions and *postconditions* specify the semantics of operations.

Box interface x *satisfies* interface y when x provides at least the operations of y and the corresponding operations of x and y have equivalent meanings.

$$pre(x,m) \Leftarrow pre(y,m) \land C(x,y) \land I(y)$$

$$x \xrightarrow{\text{extends}} y$$

$$post(x,m) \land C(x,y) \land I(x) \Rightarrow post(y,m)$$

$$\square I(x) \land C(x,y) \Rightarrow I(y)$$

Figure 5. Interface x **with operation m satisfies interface** y. I(x) is the invariant for x. pre(x,m) is the precondition for operation m on interface x. post(x,m) is the postcondition for operation m on interface x. C(x,y) is a coupling invariant that links the information models of interfaces x and y.

For a box B, let I(B) be its box invariant, C(B) be the coupling invariant that ties it to the interface information models, and prov(B) be the provided interfaces. For any box B, it must be the case that:

$$(\forall p: p \in prov(B) : I(p)) \land C(B) \Rightarrow I(B)$$

An atomic box must implement its provided interfaces as a cluster of Java classes. All of its provided interfaces must have the same information model (V,I).

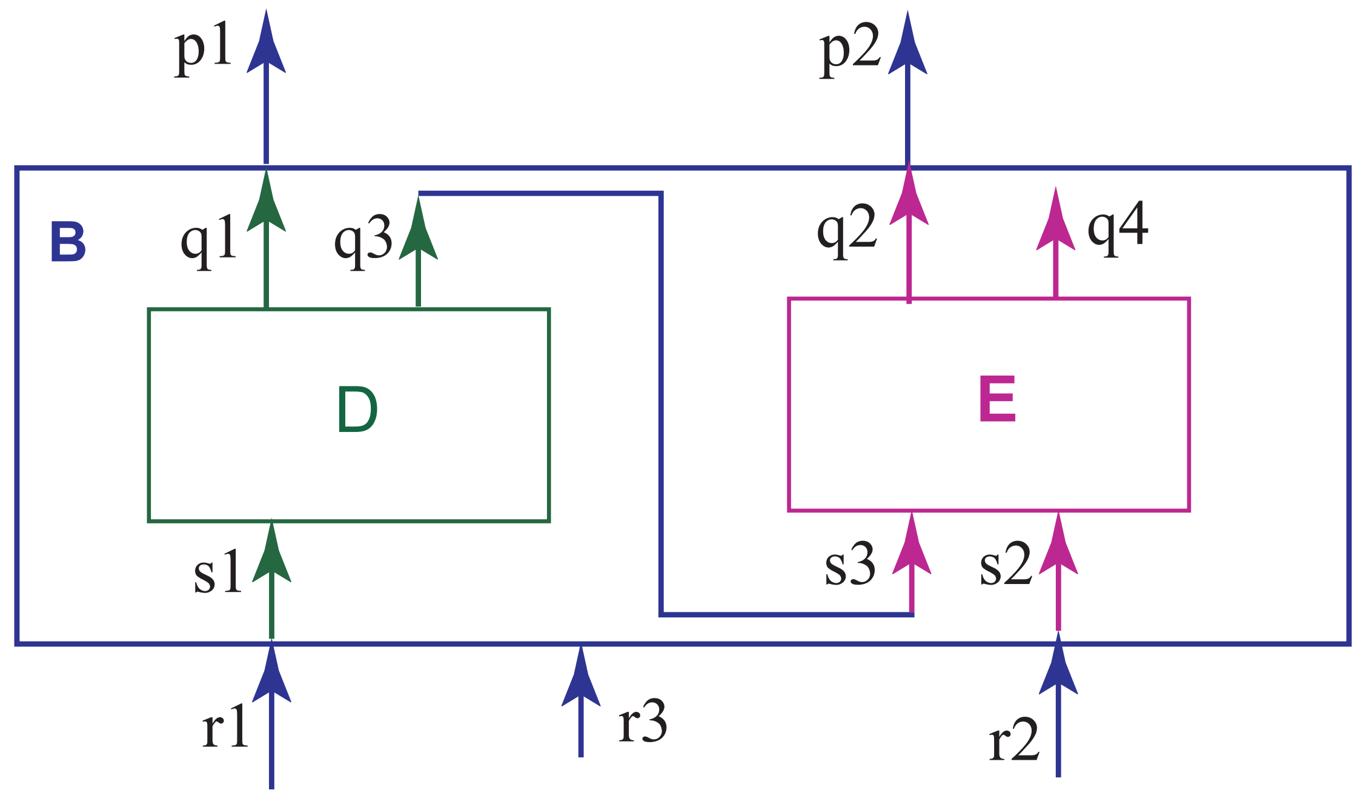

Figure 6. Compound box B **with provided interfaces** p1 **and** p2**, required interfaces** r1, r2, **and** r3, **and constituent boxes** D **and** E. If the arrow for an interfae x is linked to the one for interface y, then x must satisfy y.

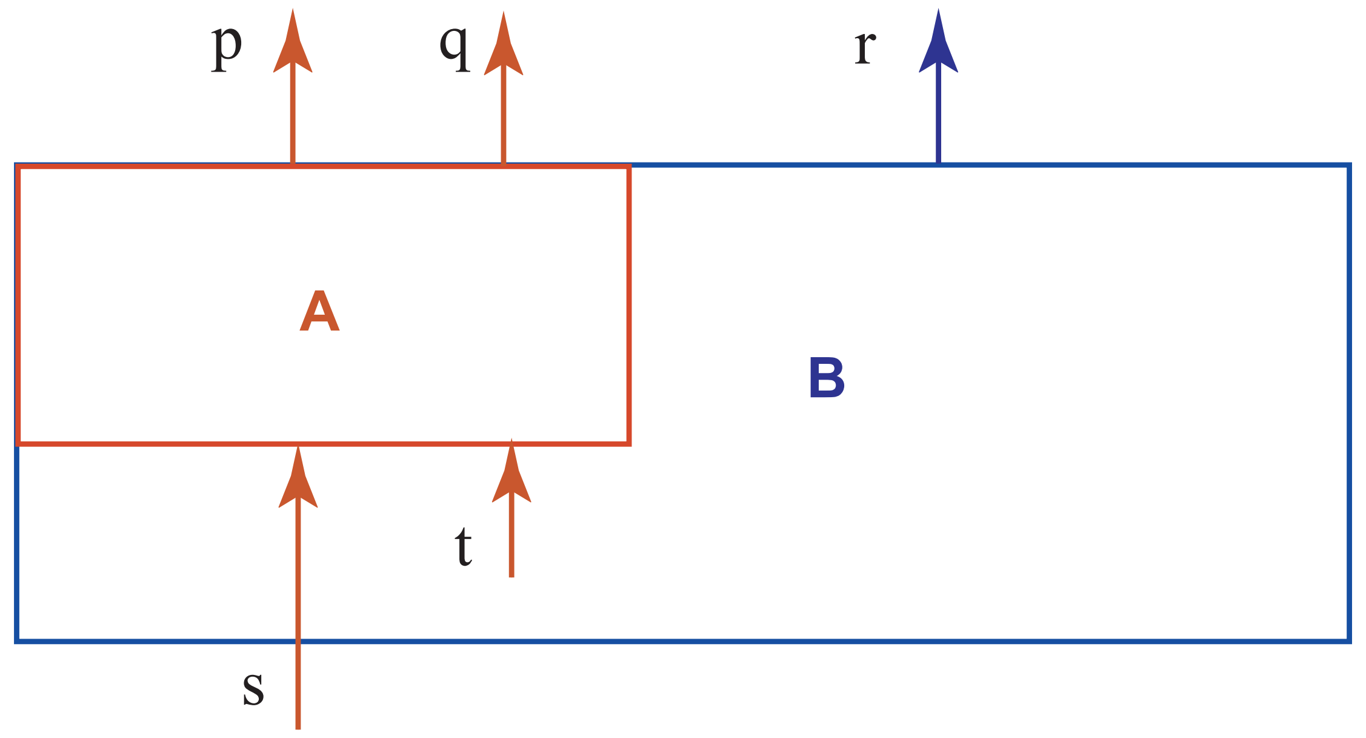

Figure 7. Box B **with provided interfaces** p, q, **and** r **and required interface** s **implements abstract box** A.

Current and Future Work

Current plans are to:

. implement BoxScript

. specify several examples

. relate to other formalisms

. integrate with JML or other tools

. investigate alternative required interface semantics (e.g., callbacks)

. develop decompostion techniques

The University of Mississippi